Romain Jacquet **BTS SIO 1**
17/12/2025

Compte rendu TP Atelier 12 : Javascript

Sommaire

Introduction

Dans cet exercice, j'ai exploré la manipulation du DOM avec JavaScript afin de mieux comprendre comment interagir avec les éléments d'une page web. L'objectif était de sélectionner des éléments HTML, de modifier leur contenu et leur style, et de parcourir la structure d'une page sans toucher directement au fichier HTML source. J'ai également étudié les relations entre les nœuds, comme `firstChild`, `lastChild`, `nextSibling` ou `previousSibling`, pour comprendre comment naviguer dans l'arborescence et identifier les éléments désirés. Enfin, j'ai appliqué ces notions pour colorer la diagonale d'un tableau et gérer dynamiquement le contenu des balises.

Navigateur 1

Exercice 1 : Modification du DOM

Dans ce premier exercice, j'ai sélectionné des éléments HTML de la page (comme un titre h1 ou une `div`) et j'ai modifié leur texte ainsi que leur style (couleur, fond) directement via JavaScript, sans toucher au fichier HTML source.

Enfants DOM

Regardez cette page :

```
<html>
<body>
  <div>Users:</div>
  <ul>
    <li>John</li>
    <li>Pete</li>
  </ul>
</body>
</html>
```

Pour chacun des éléments suivants, donnez au moins un moyen d'y accéder :

- Le noeud `<div>` du DOM ?
- Le noeud `<ul>` du DOM ?
- Le deuxième `<li>` (avec Pete) ?

La question des frères et sœurs

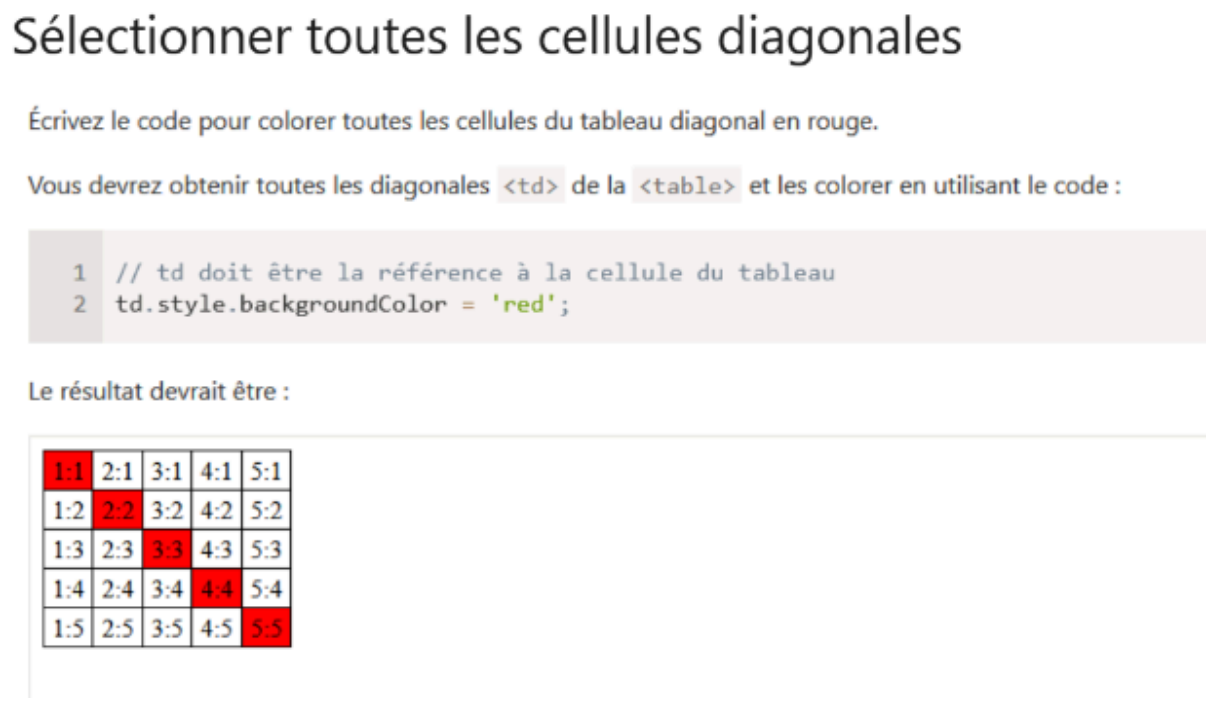
Si `element` – est un nœud élément arbitraire du DOM ...

- Est-il vrai que `elem.lastChild.nextSibling` est toujours `null` ?
- Est-il vrai que `elem.children[0].previousSibling` est toujours `null` ?

`elem.lastChild.nextSibling` est-il toujours `null` ?
Oui. Le dernier enfant d'un élément n'a, par définition, aucun nœud après lui. Il ne peut donc pas avoir de `nextSibling`.

`elem.children[0].previousSibling` est-il toujours `null` ?
Non. Même si `children[0]` désigne la première balise HTML, `previousSibling` ne se limite pas aux balises. S'il y a un espace ou un saut de

ligne avant cette balise dans le code HTML, ce sera un nœud de texte qui sera retourné, et non `null`.



Voici ci-dessous l'algorithme que j'ai utilisé pour colorer la diagonale d'un tableau en rouge.

```
let table = document.querySelector('table');

for (let i = 0; i < table.rows.length; i++) {
  let td = table.rows[i].cells[i];
  td.style.backgroundColor = 'red';
}
```

Une boucle `for` est utilisée pour parcourir les lignes du tableau. À chaque itération d'indice *i*, on accède à la cellule ayant le même indice *i*, ce qui permet de représenter visuellement la diagonale du tableau.

Recherche d'éléments

Voici le document avec le tableau et formulaire

Comment trouver ?...

1. Le tableau avec `id="age-table"`.
2. Tous les éléments `label` dans ce tableau (il devrait y en avoir 3).
3. Le premier `td` dans ce tableau (avec le mot "Age").
4. Le `form` avec `name="search"`.
5. Le premier `input` dans ce formulaire.
6. Le dernier `input` dans ce formulaire.

1. Parcours de l'arborescence DOM (nœuds enfants)

L'objectif est d'accéder aux différents éléments d'une page HTML sans recourir aux attributs `id`, en s'appuyant uniquement sur la structure du document.

Accès à la balise `<div>`
On utilise `document.body.firstElementChild`, qui correspond au premier nœud de type élément contenu dans la balise `<body>`.

Accès à la balise `<ul>`
La balise `<ul>` est obtenue grâce à `document.body.children[1]`, qui sélectionne le deuxième élément enfant du corps de la page.

Accès au deuxième élément `<li>` (Pete)
L'instruction `document.querySelectorAll('li')[1]` permet de récupérer le deuxième élément `<li>`, en tenant compte du fait que l'indexation commence à zéro.

2. Relations entre les nœuds (frères et sœurs)

Cette partie permet de mieux comprendre les relations de voisinage entre les nœuds du DOM.

`elem.lastChild.nextSibling` est toujours `null`
Vrai
Le dernier enfant d'un élément n'a pas de nœud suivant. Par conséquent, la propriété `nextSibling` renvoie toujours `null`.

`elem.children[0].previousSibling` est toujours `null`
Faux
Même si `children[0]` correspond au premier élément HTML, il peut être précédé d'un nœud texte (espaces, sauts de ligne, etc.).
Dans ce cas, `previousSibling` renverra ce nœud texte et non `null`.

3. Sélection des cellules diagonales

La boucle parcourt chaque ligne (i). Pour la ligne 0, elle colore la cellule 0, pour la ligne 1, la cellule 1, et ainsi de suite, formant une diagonale rouge.

```
let table = document.querySelector('table');

for (let i = 0; i < table.rows.length; i++) {
  table.rows[i].cells[i].style.backgroundColor = 'red';
}
```

Exemple d'utilisation des méthodes de recherche sur une structure HTML :

Récupérer le tableau par son ID :
`document.getElementById('age-table')`
Tous les labels du tableau : `table.querySelectorAll('label')`
Premier <td> (Age) : `table.querySelector('td')`
Formulaire "search" :
`document.querySelector('form[name="search"]')`
Premier champ input du formulaire : `form.querySelector('input')`
Dernier champ input du formulaire : `inputs[inputs.length - 1]`

Qu'affiche le script ?

```
1  <html>
2
3  <body>
4    <script>
5      alert(document.body.lastChild.nodeType);
6    </script>
7  </body>
8
9  </html>
```

1. Compter les descendants : que montre le script ?
 Le script renvoie 1 pour document.body.lastChild.nodeType.

Explication simple :
 Quand le navigateur lit la page, il va de haut en bas. Au moment où le script s'exécute, il est le dernier élément du <body>.
 lastChild renvoie donc… le script lui-même.
 Et pour tout élément HTML, nodeType vaut 1.

En gros : le script se dit "Hey, je suis un élément HTML !" → d'où le 1.

Balise dans le commentaire

Qu'affice ce code ?

```
<script>
  let body = document.body;

  body.innerHTML = "<!--" + body.tagName + "-->";

  alert( body.firstChild.data ); // Qu'est ce qu'il y a ici ?
</script>
```

Où est le "document" dans la hiérarchie ?

À quelle classe appartient le `document` ?

Quelle est sa place dans la hiérarchie DOM ?

Hérite-t-il de `Node` ou `Element`, ou peut-être de `HTMLElement` ?

1. Compter les descendants

Question : Qu'affiche le script ?
 Réponse : 1

Pourquoi ?
 Le script est **le dernier élément du <body>** au moment où il s'exécute.
 lastChild pointe donc sur lui-même.
 Comme toutes les balises HTML ont un nodeType égal à 1, l'alerte affiche 1.

2. Balise dans le commentaire

Question : Qu'affiche ce code ?
Réponse : BODY

Pourquoi ?

- body.innerHTML = "<!--BODY-->" remplace tout le contenu par un **commentaire**.
- body.firstChild devient ce commentaire.
- .data récupère le texte à l'intérieur du commentaire.

Donc l'alerte affiche BODY.

3. À propos du document

- **Classe :** HTMLDocument (ou Document). C'est l'objet qui représente **toute la page**.
- **Position dans le DOM :** c'est la **racine**, le sommet de l'arbre. On passe par lui pour accéder aux éléments.
- **Héritage :** il hérite de Node, mais pas de Element ni HTMLElement. C'est le conteneur global, pas une balise individuelle.

Navigateur 2

Écrivez le code pour sélectionner l'élément avec l'attribut `data-widget-name` dans le document et pour lire sa valeur.

```
<!DOCTYPE html>
<html>
<body>

  <div data-widget-name="menu">Choose the genre</div>

  <script>
    /* your code */
  </script>
</body>
</html>
```

```
let widget = document.querySelector('[data-widget-name]');
let name = widget.dataset.widgetName;
console.log(name);
```

```
let widget = document.querySelector('[data-widget-name]');
let name = widget.getAttribute('data-widget-name');
console.log(name);
```

Rendre les liens externes orange

Mettez tous les liens externes en orange en modifiant leur propriété `style`.

Un lien est externe si :

- Son `href` contient `://`
- Mais ne commence pas par `http://internal.com`.

Exemple :

```
<a name="list">the list</a>
<ul>
  <li><a href="http://google.com">http://google.com</a></li>
  <li><a href="/tutorial">/tutorial.html</a></li>
  <li><a href="local/path">local/path</a></li>
  <li><a href="ftp://ftp.com/my.zip">ftp://ftp.com/my.zip</a></li>
  <li><a href="http://nodejs.org">http://nodejs.org</a></li>
  <li><a href="http://internal.com/test">http://internal.com/test</a></li>
</ul>

<script>
  // setting style for a single link
  let link = document.querySelector('a');
  link.style.color = 'orange';
</script>
```

Le résultat devrait être :

The list:

- http://google.com
- /tutorial.html
- local/path
- ftp://ftp.com/my.zip
- http://nodejs.org
- http://internal.com/test

```
let links = document.querySelectorAll('a');

for (let link of links) {
  let href = link.getAttribute('href');
  if (href && href.includes('://') && !href.startsWith('http://internal.com')) {
    link.style.color = 'orange';
  }
}
```

createTextNode vs innerHTML vs textContent

Nous avons un élément DOM vide `elem` et une chaîne de caractères `text`.

Lesquelles de ces 3 commandes feront exactement la même chose ?

1. `elem.append(document.createTextNode(text))`
2. `elem.innerHTML = text`
3. `elem.textContent = text`

`elem.append(document.createTextNode(text))`
Cette méthode ajoute le texte exactement tel qu'il est dans la page.
Tout ce que contient `text` — même des balises HTML comme `<b>` ou `<script>` — sera affiché comme du texte normal, sans être interprété par le navigateur.
En clair : ce que tu écris, c'est exactement ce que l'utilisateur voit.

`elem.innerHTML = text`
Celle-ci fonctionne différemment : elle dit au navigateur de lire le texte comme du code HTML.
Par exemple, si `text` contient `<b>Salut</b>`, le mot Salut s'affichera en gras.
Attention : si le texte provient d'une source non fiable, cela peut créer une faille de sécurité (XSS).

`elem.textContent = text`
C'est la version moderne et simplifiée de la première méthode.
Elle met le texte dans l'élément sans jamais interpréter les balises HTML.
Le résultat est identique à `createTextNode`, mais plus simple à utiliser.

```html
<ol id="elem">
  <li>Hello</li>
  <li>World</li>
</ol>

<script>
  function clear(elem) { /* votre code */ }

  clear(elem); // efface la liste
</script>
```

```
function clear(elem) {
  elem.replaceChildren();
}
```

*

Pourquoi "aaa" reste-t-il ?

Dans l'exemple ci-dessous, l'appel `table.remove()` supprime le tableau du document.

mais si vous l'exécutez, vous pouvez voir que le texte `"aaa"` est toujours visible.

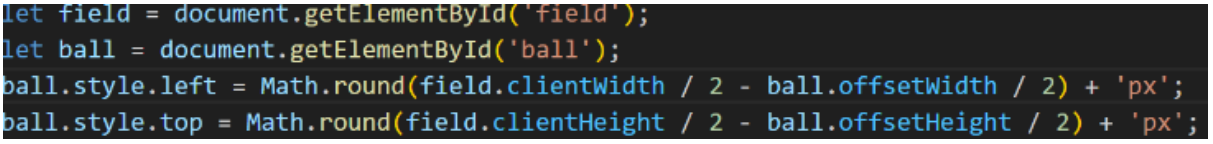
Pourquoi cela se produit-il ?

```html
<table id="table">
  aaa
  <tr>
    <td>Test</td>
  </tr>
</table>

<script>
  alert(table); // la table, comme il se doit

  table.remove();
  // pourquoi y a-t-il encore "aaa" dans le document ?
</script>
```

Pour vider un élément HTML, il y a plusieurs façons : tu peux simplement lui donner un contenu vide pour tout supprimer d'un coup, utiliser une fonction moderne qui remplace tous ses enfants par rien pour une approche plus propre, ou encore parcourir ses enfants et les retirer un par un en commençant toujours par le premier, car la liste se met à jour à chaque suppression.

Le texte "aaa" reste visible même après avoir supprimé un tableau parce que les navigateurs corrigent automatiquement les erreurs HTML : un tableau ne peut contenir que des lignes ou des cellules, donc le texte placé directement dedans est déplacé juste avant le tableau pour ne pas casser l'affichage. Quand tu supprimes le tableau, ce texte est déjà en dehors et reste affiché.

```
function getScrollbarWidth() {
  const outer = document.createElement('div');
  outer.style.visibility = 'hidden';
  outer.style.overflow = 'scroll';
  outer.style.msOverflowStyle = 'scrollbar';
  document.body.appendChild(outer);

  const inner = document.createElement('div');
  outer.appendChild(inner);

  const scrollbarWidth = (outer.offsetWidth - inner.offsetWidth);

  outer.parentNode.removeChild(outer);

  return scrollbarWidth;
}

console.log("La largeur de la barre de défilement est : " + getScrollbarWidth() + "px");
```



```
let field = document.getElementById('field');
let ball = document.getElementById('ball');
ball.style.left = Math.round(field.clientWidth / 2 - ball.offsetWidth / 2) + 'px';
ball.style.top = Math.round(field.clientHeight / 2 - ball.offsetHeight / 2) + 'px';
```

Conclusion

Cet exercice m'a permis de mieux comprendre le DOM et comment accéder ou modifier ses nœuds. J'ai appris à utiliser `textContent`, `innerHTML` et `append` selon le type de contenu, et à différencier les nœuds texte, les commentaires et les éléments HTML. J'ai aussi vu que les navigateurs corrigent automatiquement certaines erreurs HTML. Ces connaissances sont importantes pour créer des pages web dynamiques et prévoir le comportement des scripts.